FOCS Day 16 In Class 1

1. Algorithm A

Consider the following pseudo-[1] pseudocode:

```
1. create a sequence that contains only node a
2. until the collection is empty:
3.  remove node n from the head of the sequence and visit it
4.
5. to visit a node:
6.  add unvisited adjacent nodes to the tail of the sequence
```

(a) What data type is ***sequence***?

(b) Run this algorithm on the provided graphs. In what order does it visit the nodes? For each graph, list the nodes in the order visited.

For a *directed graph*, replace “adjacent nodes” by “direct successors”. For example, in (ii), the direct successors of node *a* are nodes *b* and *c*; the direct successors of node *b* are nodes *d* and *e*.

(c) Will this algorithm always visit all the vertices in a graph? Provide a proof sketch or a counter example.

(d) What will happen if if the word “unvisited” is removed from line 5?

(e) How many times does algorithm A visit each node? Each edge? Try to write an equation in terms of the number of nodes (*v*) and edges (*e*).

---

[1] This is even *less* detailed (higher level) than pseudo-code. It therefore makes you (the reader) do more of the work in order to execute or implement it; hides sources of complexity; and hides sources of unfeasibility.

2. Algorithm B

Modify Algorithm A to add nodes to the *head* of the sequence instead of the *tail* (line 6). This is Algorithm B.

```
1. create a sequence that contains only node a
2. until the sequence is empty:
3.  remove node n from the head of the sequence and visit it
4.
5. to visit a node:
6.  add unvisited adjacent nodes to the head of the sequence
```

(a) What data type is ***sequence***?

(b) Run this algorithm on the provided graphs. In what order does it visit the nodes? For each graph, list the nodes in the order visited.

(c) How many times does algorithm B visit each node? Each edge?

3. Algorithm C

```
1. visit node a
2.
3. to visit a node:
4.   visit its unvisited adjacent nodes (graph) / successors (digraph)
```

Compare Algorithm C to Algorithms A and B.

(a) Is it functionally similar?

(b) How does it compare with respect to time and space?

4. Algorithm D

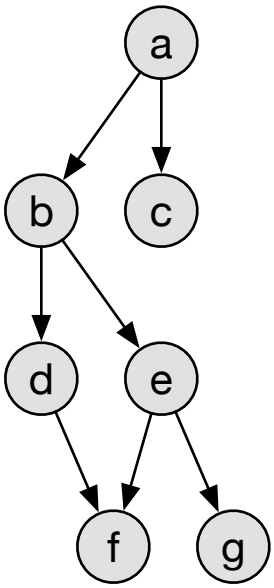
In a directed graph, a node's ***indegree*** is the number of arrows that land on it. Its ***outdegree*** is the number of arrows that emanate from it. A ***root*** is a node with zero indegree. A ***leaf*** is a node with zero outdegree. [2]

(a) Do the definitions of root and leaf for a *digraph* agree with the definitions of root and leaf for a *tree*?

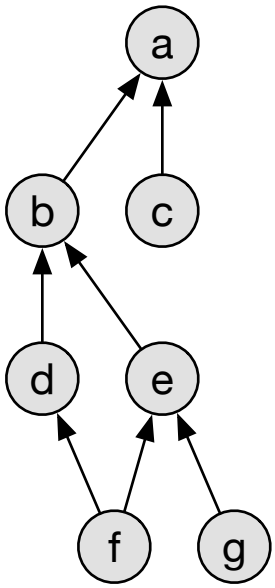
(b) Consider Algorithm D (below).

```
1. while there's a root n:
2.   visit n
3.
4. to visit a node n:
5.   remove n and its edges from the graph
```

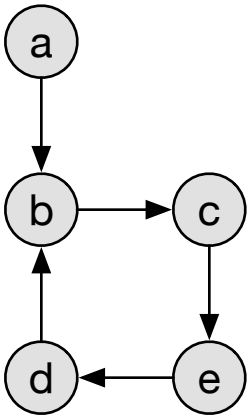
In what order does Algorithm D visit the nodes in each of these trees? Note that line 5 may create new roots.



(i)



(ii)



(iii)

(c) Think about actually implementing Algorithm D. How much duplicate effort is it doing? Can you improve on this? [This is a vaguely worded question. We'll discuss this in class.]

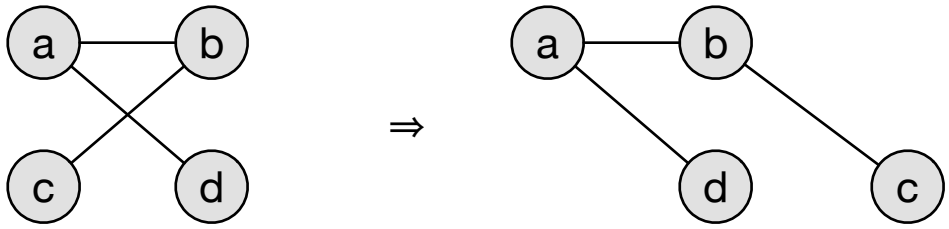
---

[2] *Indegree*, *outdegree*, *root*, and *leaf*, (with the inter-word spaces omitted as shown) are standard terms of graph theory. *Land* and *emanate* are not.

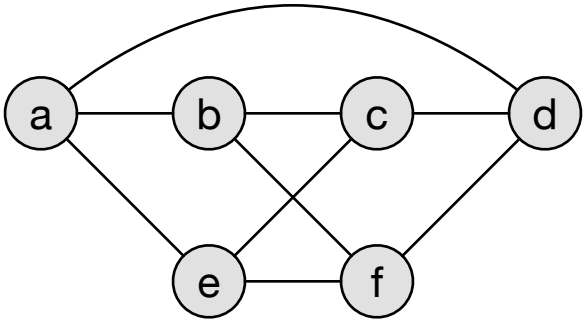
(Optional) Challenge 1

A planar embedding is a diagram of a graph, that doesn't have any crossing edges. A planar graph is a graph that has a planar embedding.

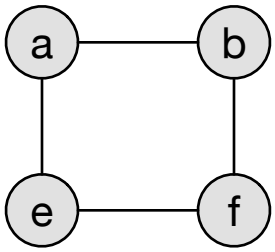
The diagram on the left is not a planar graph, because it has edges that cross. The graph that it's a diagram of, however, is a planar graph, because there exists a planar embedding (because it's possible to move the vertices around to remove the edge crossings).



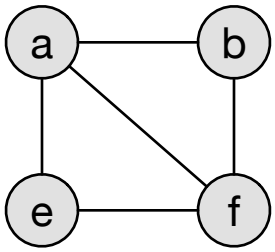
This diagram is not a planar graph. Is there a planar embedding of the same graph?



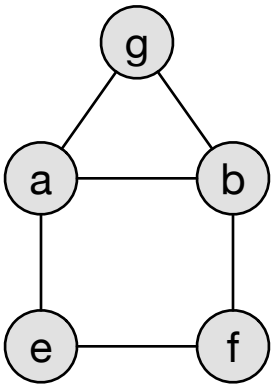
More reading (and spoiler): Once you've tried this, look up the *Three Utilities Problem* in Wikipedia to see this problem's history, some wording you can use to pose it without talking about graphs, and the answer.



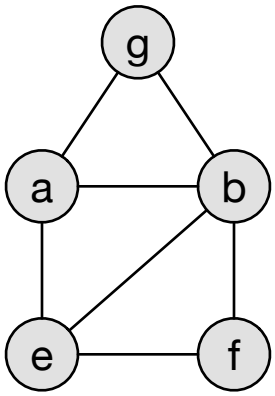
(i)



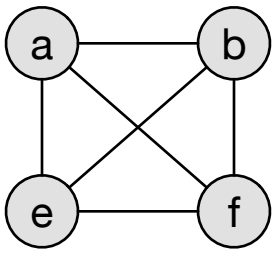
(ii)



(iii)



(iv)



(v)

(Optional) Challenge 2

It is possible to traverse all the edges in graph (i) without using any edge twice. Is it possible to do so with the other graphs? What characterizes the difference?

(a) It is possible to traverse all the edges in graph (i) without using any edge twice.

(b) Is it possible to do so with the other graphs? What characterizes the difference?

(c) More reading: Search Wikipedia for ***Eulerian cycle***, ***Eulerian path***